"Learn at least a half dozen programming languages. Include one language that emphasizes class abstractions (like Java or C++), one that emphasizes functional abstraction (like Lisp or ML or Haskell), one that supports syntactic abstraction (like Lisp), one that supports declarative specifications (like Prolog or C++ templates), and one that emphasizes parallelism (like Clojure or Go)."

Peter Norvig "Teach Yourself Programming in Ten Years"
`http://norvig.com/21-days.html`

CPSC 312 — Functional and Logic Programming

- Professor: David Poole
- URL: `http://www.cs.ubc.ca/~poole/cs312/2024/`
  Assignment 1 is due next Tuesday!
- We will use Canvas for assignment submission, grades and Zooming. Classes will be simulcast on Zoom and also recorded an available on Canvas. (You may have to remind me to record).
- Every student should expect to struggle, but can succeed! You learn by doing and making mistakes.
- Ask questions!
- Quote of the day: "Apparently, the university is the only place where you pay for something, and then try as hard as you can NOT to get your money's worth."
  (`https://people.cs.kuleuven.be/~bart.demoen/`)

CPSC 312 — Assessment

- Marks:
  - 40%: 2 projects (groups 2 or 3) with demos
  - 30%: 3 midterms (10% each). Tentative dates on web page (subject to change).
  - 25%: final exam
  - 3%: assignments (marked for participation)
  - 2% informative discussion posts.

Estimates:

- Everyone can pass
- If you memorize and can reproduce everything presented in class you can get a B- or C+.
- For an A or A+ you have to "get it" ("aha!" moment).

Clicker Question

I am taking CPSC 312 because (pick best answer)

A I want to learn as many programming paradigms as possible

B Haskell and Prolog programmers make lots of money

C I am fascinated by the ideas of functional and/or logic programming

D I heard that 312 is an easy course

E I just need another (3rd year) course

Lecture Overview

- What is logic and functional programming?
- Simple Haskell programs and queries.

Learning objectives: at the end of the class, you should be able to

- recognize syntax and semantics of Haskell
- write a simple Haskell program

What is functional and logic programming?

- Program is a high-level specification of what should be computed, not how it should be computed.
- Try to find representations that are as close to the problem domain as possible
- Abstract away from the state of a computer
- Programming and debugging should all be questions about the domain, not about the computation.
- Allow computer to decide how to most efficiently implement the program.
- To solve a complex problem, break it into simpler problems.
- Variables cannot change their values. Controlled side effects.
- Haskell is a strongly typed language. You don't need to declare types. Type checking is done at compile time.

Choosing a Representation Language

We need to represent a problem to solve it on a computer.

$$\left[\begin{array}{l}\text{problem} \\ \quad \rightarrow \ \text{specification of problem} \\ \qquad \rightarrow \ \text{appropriate computation}\end{array}\right]$$

Example specification languages: Machine Language, C++, Java, Haskell, Prolog, English

Haskell

Haskell lets one:

- evaluate expressions
- define functions

`http://cs.ubc.ca/~poole/cs312/2024/haskell/First.hs`

Syntax

- comments are either
  `--` comment to end of line or
  `{-` comment `-}`
- variables either:
  - prefix: made up of letters, digits, ' or _ and start with a lower-case letter
  - infix: made up of sequences of other characters
- indentation is significant
- parentheses are used for precedence and tuples (not for arguments of functions)
- Function application binds most strongly
  `fac 3*5` means
  `(fac 3)*5`
- Binary prefix functions can be made infix using back-quotes, e.g. `` `div` ``
  Infix operators can be made prefix using parentheses, e.g. (*)

Clicker Question

Which of the following is not true:

A Haskell functions require parentheses (like Java and C)

B Haskell variables cannot change their values

C Haskell is a strongly typed language

D You don't need to declare the types of all functions

Clicker Question

Which is the true of the expression:
`foo bar zoo`

A `foo` must be a function

B `bar` must be a function

C `bar` cannot be a function

D `zoo` must be a number

E `bar` and `zoo` must be of the same type

Clicker Question

Which is the true of the expression:
`foo @#$%^& zoo`

A `foo` must be a function

B `@#$%^&` must be a function

C `@#$%^&` cannot be a function

D `zoo` must be a number

E `foo` must not be a function

Definition of a function

- Function Definition:

  `name x1 x2 ... xk = e`

  `x1 x2 ... xk` are formal parameters
  `e` is an expression
- `xi` can contain structures, but each variable can only appear once.
- Multiple equations can define a function; the first one to succeed is used.

Evaluation of Haskell program

- Haskell evaluates expressions.
- Haskell knows how to implement some expressions (such as 3+4*7)
- Given the definition of name:

  `name x1 x2 ... xk = e`

  The expression

  `name v1 v2 ... vk`

  when all `k` arguments are provided evaluates to value of
  `e {x1/v1, x2/v2, ..., xk/vk}`
  which is same as `e` but with each `xi` replaced with `vi`
- `foo x y = 1000*x+y`
  `foo 9 3`
  `x*1000+y {x/9, y/3}` evaluates to value of 9*1000+3
  which is 9003.